YoYoLint – a custom SystemVerilog RTL linter for Yosys

Ajeetha Kumari Venkatesan
Director of Verification
AsFigo Technologies
London, UK
ajeethak@asfigo.com

Deepa Palaniappan
ASIC *DAV Engineer*
AsFigo Technologies
Rapperswil, Switzerland
deepa@asfigo.com

Saanvi Pradhan
AsFigo Technologies
High School Senior, LTHS
Bee Cave, Texas USA
saanvi.pradhan@gmail.com

***Abstract*— YoYoLint is an open-source static analysis tool designed for SystemVerilog RTL code. Written in Python and built around Slang/PySlang YoYoLint derives learnings from PySlint, a popular SystemVerilog testbench linter. YoYoLint aims to assist users in ensuring their SystemVerilog code is well-prepared for synthesis with Yosys, a popular, opensource RTL synthesis tool. YoYoLint helps users prepare SystemVerilog code for synthesis with Yosys, an open-source RTL synthesis tool. It provides a set of configurable linting rules that focus on detecting code patterns and issues that might cause problems during synthesis with Yosys.**

The tool runs efficiently and delivers clear feedback on code violations. Users can run YoYoLint before synthesis with Yosys to find and fix issues, improving code quality and compatibility.

YoYoLint is extendable, allowing users to add custom linting rules. It is regularly updated by analyzing SystemVerilog designs with Yosys, incorporating new unsupported features into its rule set. This ensures YoYoLint remains useful for current coding practices.

The paper also addresses the role of linters in managing coding styles and avoiding common pitfalls in SystemVerilog. It provides an overview of YoYoLint's development, potential extensions, and initial results.

We share our experience in building YoYoLint and potential extensions. We conclude with the results on a few early use cases.

Keywords—SystemVerilog, PySlint, YoYoLint, Yosys, RTL, Synthesis, Linting

I. INTRODUCTION

RTL linting is a static analysis technique used to identify potential issues in Register Transfer Level (RTL) code before synthesis. By examining the code without executing it, linting tools can detect errors, inconsistencies, and problems that could lead to unexpected behavior or synthesis failures.

Yosys is a widely-used open-source RTL synthesis tool that converts Verilog, SystemVerilog, or VHDL code into a gate-level netlist. It offers a flexible TCL (Tool Command Language) interface, allowing users to control the tool's behavior through scripting. This makes Yosys suitable for both interactive use and automation tasks. While Yosys fully supports Verilog, its SystemVerilog support is partial, focusing primarily on synthesizable constructs. Advanced features like assertions and complex data types are not fully supported.

SystemVerilog extends Verilog by introducing enhanced data types, complex procedural constructs, and built-in verification features. These additions make SystemVerilog ideal for designing and modeling digital systems at both the behavioral and RTL levels. However, the complexity of SystemVerilog can pose challenges when transitioning from RTL code to gate-level implementations in synthesis tools like Yosys.

To ensure successful synthesis of SystemVerilog designs in Yosys, it is crucial to adhere to specific coding guidelines and avoid common pitfalls. Proper attention to synthesizable constructs and coding practices is key to achieving compatibility with synthesis tools.

II. BUILDING CUSTOM LINTERS

Some aspects of coding styles are often choices. In a complex language like SystemVerilog, which is widely used in the ASIC and FPGA design industries, there are multiple ways to achieve a given design or testbench. To ensure code consistency and adherence to team-specific styles, many teams use static checkers (linters). Linters also help to identify common pitfalls, such as unintended functionality or race conditions.

While linting has long been used in the industry, open-source options have historically been limited. However, with the recent availability of open-source parsers, innovative linting solutions have emerged. PySlint is one such linter, developed on top of the open-source Slang/PySlang parser. It enables users to create custom rules to check SystemVerilog testbenches using a Python API. Inspired by the momentum around PySlint, we developed YoYoLint to focus specifically on ensuring compatibility with Yosys.

III. SYSTEMVERILOG SUPPORT IN YOSYS

The open-source version of Yosys officially supports Verilog 2005. However, unofficial modifications have extended its capabilities to include several SystemVerilog features. These additions encompass advanced constructs like enums and typedefs, which are part of SystemVerilog. One of the challenges that we face often with Yosys is that there is no

well defined subset of SystemVerilog R|TL supported by Yosys. Often Yosys throws non-descriptive errors messages on unsupported features in SystemVerilog. We have captured a few of these limitations below.

A. *Two state data type in ports*

SystemVerilog's int data type with a two-state representation is not supported for use in ports within Yosys.

B. *MDA support*

Yosys does not support multidimensional arrays (MDAs). This limitation means that designs using MDAs cannot be directly synthesized with Yosys. Users must work around this restriction by converting MDAs into supported data structures or flattening the arrays.

C. *Parameters of type int*

SystemVerilog parameters of type *int* are not yet supported in Yosys. For instance, parameters declared with *int* cannot be utilized, and users should opt for alternative types like *integer* for compatibility with Yosys.

D. *Streaming operators*

Yosys does not support streaming operators, which are used for operations like bit concatenation and repetition. This limitation means that constructs involving such operators cannot be synthesized with Yosys. To work around this, users will need to use alternative methods to achieve similar functionality in their designs.

E. *Importing package*

Basic package is supported. However, importing does not work. Fully qualified package references work in Yosys.

F. *User defined typedef*

Yosys does not support *typedef* in packages

G. *Adaptability to Different Coding Styles and Projects*

Different teams and projects often adhere to their own coding styles and conventions. A configurable YoYoLint allows users to tailor the rules and checks to their specific needs.

H. *SVA support in Yosys*

Yosys supports many advanced features of SystemVerilog Assertions (SVA), such as:

- Default clocking and endclocking declarations

- Default disable conditions

- Definition and use of properties and sequences

- Definition and application of checkers

- Handling of arguments within sequences, properties, and checkers

- Organization of sequences, properties, and checkers within packages

Additionally, Yosys supports SystemVerilog's `bind` statement and deep hierarchical references, facilitating the integration of formal properties with the design under test.

IV. Implementation

We use Python to implement these lint checks in YoYoLint. We derive heavily from PySlint, which is a static analysis tool for SystemVerilog testbenches built on top of the Slang/PySlang parser. It leverages the capabilities of these parsing libraries to analyze SystemVerilog code and identify potential issues. The SystemVerilog code is parsed into an abstract syntax tree (AST). The AST is a hierarchical representation of the code's structure, providing information about modules, declarations, statements, and expressions.

Below is a sample rule implementation:

```
def INT_PARAM_NYS(lvDecl):
  if (lvDecl.kind.name ==
'ParameterDeclarationStatement'):
    lvRID = 'INT_PARAM_NYS'
    lv_dt_s = str(lvDecl.parameter.type).strip()
    if (lv_dt_s == 'int'):
      msg = 'SystemVerilog parameter of type int is '
      msg += 'NYS - Not Yet Supported in Yosys'
      msg += str(lvDecl)
      yYLMsg (lvRID, msg)
```

As it can be seen, it is not a regexp based pattern matching to find unsupported constructs in user's code. Rather we use a solid data model built on the lines of Verilog's VPI (Verilog Procedural Interface a.k.a. PLI).

We use a common messaging API named *yYMsg*, a sample implementation of the same is below:

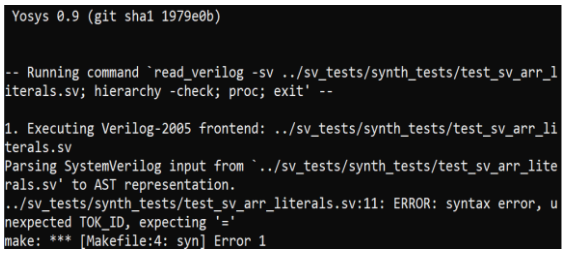
```
def yYLMsg(rule_id, msg):
  if (yYLRenabled(rule_id)):
    lvYylStr = 'yoYoLint: Violation: ['
    lvYylStr += rule_id
    lvYylStr += ']: '
    lvYylStr += msg
    print(lvYylStr)
```

As it can be noted in the API above, each message can be enabled/disabled providing finer control to the user. Much of these features are borrowed from PySlint.

V. Benefits

YoYoLint provides several benefits as a linter for SystemVerilog RTL in conjunction with Yosys. It offers quick identification of issues in SystemVerilog code that may affect synthesis, such as unsupported features or potential syntax errors. The tool generates clear, actionable messages, helping users address problems before running Yosys. Additionally, YoYoLint is extendable, allowing users to add custom rules based on their specific needs and continuously improving support for SystemVerilog features.

For instance, below is an error message from Yosys for an unsupported feature:



While Yosys is great in its value in terms of synthesis, mapping, optimizations and generating bit streams etc. the parser level errors are not the real focus of the tool. This is precisely where YoYoLint fits in nicely.

VI. Conclusion

We present our contribution to YoYoLint for SystemVerilog RTL linting, focusing on enhancing configurability. This improvement allows users to tailor linting rules to better fit various SystemVerilog design projects, promoting greater adaptability and usability. The solution not only addresses the current need for customizable linting but also lays the groundwork for future developments and extensions in YoYoLint. The paper concludes with considerations for future research, including the exploration of additional configuration parameters for specific YoYoLint rules.

Acknowledgments

We sincerely thank our mentor Ajeetha at AsFigo and various team members who have helped us with running YoYoLint on various designs and giving us valuable feedback.

References

[1] IEEE 1800 SystemVerilog LRM
[2] PySlint repository: https://github.com/ASFigo/PySlint
[3] PySlang - https://pypi.org/project/pyslang/
[4] Deepa Palaniappan – PySlint paper at ORConf 2023, Munich, Germany
[5] TOML – https://toml.io
[6] tinyODIN https://github.com/ChFrenkel/tinyODIN
[7] OpenTitan: https://github.com/lowrisc/opentitan
[8] YoYoLint repository: https://github.com/AsFigo/yoYoLint